

Refatoração de Código em Kotlin

Autores: João Vitor da Conceição de Almeida Deivid
Souza dos Santos Oliveira Flávio André Almeida
Gomes Neto

Feira de Santana, 2025

Agenda

O objetivo é apresentar o resultado da refatoração e estruturação do sistema.

1. Contextualização:

Refatoração do código e estruturação do Sistema e implementação de melhorias para manter a legibilidade

2. Descrição da Atividade:

Detalhando a tarefa principal: Estruturação e refatoração de um Sistema de Pedidos em Kotlin via console

3. Estrutura do Sistema

Desenvolvido:

Apresentação do menu principal, itens, pedidos e fluxo de funcionamento

Contextualização



- Processo de revisar e reorganizar o código sem alterar sua funcionalidade.
- Torna o código mais limpo, legível e fácil de manter.
- Reduz complexidade e elimina redundâncias.
- Prepara o sistema para crescimento e novas funcionalidades.
- Aumenta a consistência e reduz riscos de erros.

Descrição da Atividade



- Melhorar organização, clareza e escalabilidade do código.
- Arquitetura procedural e funcional, sem orientação a objetos.
- Estruturação dos dados com data class
- Consistência com enum class
- Separação de responsabilidades
- Código dividido em dois arquivos:
- Model.kt - lógica de negócio, data class, funções e enum de status.
- Main.kt - interação com o usuário, menus e execução do programa.

Estrutura do Sistema



Dividido em dois arquivos.

Model.kt: Nucleo da apresentação

```
enum class OrderStatus {  
    ACCEPTED,  
    IN_PROGRESS,  
    DONE,  
    WAITING_DELIVERY,  
    OUT_FOR_DELIVERY,  
    DELIVERED  
}  
  
data class Item(  
    val code: Int,  
    val name: String,  
    val description: String,  
    val price: Double,  
    val amount: Int  
)  
|  
data class Order(  
    val code: Int,  
    var status: OrderStatus,  
    val total: Double,  
    val items: MutableList<Item>,  
    val hasDiscount: Boolean  
)
```

- Foi utilizado a estrutura de enumclass para representar valores constantes que é ideal para representar estados ou categorias, no nosso caso o primeiro.
- Foi utilizado também data classes que são perfeitas para representar entidades simples como itens e pedidos pelo fato de gerar funcionalidades automaticas.

Estrutura do Sistema

Model.kt: Núcleo da apresentação



```
val items = mutableListOf<Item>()
val orders = mutableListOf<Order>()

fun registerItem(code: Int, name: String, description: String, price: Double, amount: Int) {
    items.add(Item(code, name, description, price, amount))
}

fun updateItem(code: Int, name: String, description: String, price: Double, amount: Int): Boolean {
    val index = items.indexOfFirst { it.code == code }
    return if (index != -1) {
        items[index] = Item(code, name, description, price, amount)
        true
    } else false
}

fun createOrder(code: Int, orderItems: MutableList<Item>, hasDiscount: Boolean): Order {
    var total = orderItems.sumOf { it.price }
    if (hasDiscount) total *= 0.9
    val order = Order(code, OrderStatus.ACCEPTED, total, orderItems, hasDiscount)
    orders.add(order)
    return order
}
```

- **Listas globais:**
- items → armazena itens do cardápio
- orders → armazena pedidos
- **Funções principais:**
- registerItem → cadastra novo item
- updateItem → atualiza item existente (retorna sucesso ou falha)
- createOrder → cria pedido, calcula total e aplica desconto

Refatoração de Código em Kotlin

Estrutura do Sistema



- Funções auxiliares:

Para uma experiência de usuário robusta e sem erros, implementamos funções auxiliares de leitura.

```
package app.view// Model.kt
import app.model.*

//auxiliary functions
fun readPrice(prompt: String): Double {
    var price: Double
    do {
        print(prompt)
        val input = readln().replace(",", ".")
        price = input.toDoubleOrNull() ?: -1.0
        if (price <= 0.0) println("Invalid value. Enter a price greater than zero.")
    } while (price <= 0.0)
    return price
}
```

Refatoração de Código em Kotlin

- **Objetivo:** ler e validar um preço digitado pelo usuário.
- **Tratamento:**
 - Substitui vírgula por ponto (aceita diferentes formatos decimais).
 - Converte para número (Double).
 - Rejeita valores inválidos ou menores/iguais a zero.
- **Validação:** repete a solicitação até receber um valor válido.

Refatoração de Código em Kotlin

Estrutura do Sistema



Main.kt: ponto de entrada da aplicação

```
fun main() {  
    var itemCodeGenerator = 0  
    var orderCodeGenerator = 0  
    var option: Int  
  
    do {  
        println("=====")  
        println("                MENU                ")  
        println("1. Register Item")  
        println("2. Update Item")  
        println("3. Create Order")  
        println("4. Update Order")  
        println("5. Consult Orders")  
        println("0. Exit")  
        println("=====")  
  
        option = readOption("Choose an option: ", 0..5)
```

- Controla o fluxo com **loop** do...while.
- **Geradores de código**
- itemCodeGenerator → códigos únicos para itens.
- orderCodeGenerator → códigos únicos para pedidos.
- **Menu principal**
- Registrar Item
- Atualizar Item
- Criar Pedido
- Atualizar Pedido
- Consultar Pedidos
- Sair
- **Leitura da opção**
- Função readOption garante entradas válidas (0 a 5).
- **Fluxo de execução**
- Usuário escolhe opção → chama funções do **Model.kt**.
- Programa roda até opção **0 (Exit)**.

Estrutura do Sistema



Registrar item

```
when (option) {  
    1 -> {  
        do {  
            print("Product name: ")  
            val name = readln()  
            print("Description: ")  
            val description = readln()  
            val price = readPrice("Price: ")  
            val amount = readAmount("Stock quantity: ")  
  
            itemCodeGenerator++  
            registerItem(itemCodeGenerator, name, description, price, amount)  
            println("Item registered successfully, code: $itemCodeGenerator")  
        } while (readYesNo("Register another item? (Y/N): "))  
    }  
}
```

Separação e estruturação do Sistema via console em Klotin

Estrutura do Sistema



Atualizar item

```
2 -> {
    if (items.isEmpty()) {
        println("No items registered.")
    } else {
        items.forEach {
            println("Code: ${it.code}, Name: ${it.name}, Price: ${"%0.2f".format(it.price)}, " +
                "Stock: ${it.amount}")
        }
        val code = readExistingCode("Enter the item code to update: ",
            items.map { it.code })

        print("New name: ")
        val name = readln()
        print("New description: ")
        val description = readln()
        val price = readPrice("New price: ")
        val amount = readAmount("New stock quantity: ")

        if (updateItem(code, name, description, price, amount)) {
            println("Item updated successfully.")
        } else {
            println("Update failed. Item not found.")
        }
    }
}
```

Refatoração de Código em Kotlin

Considerações finais



- A refatoração foi essencial para transformar o protótipo em uma base sólida e escalável.
- A separação entre Model.kt (lógica de negócio) e Main.kt (interface com o usuário) trouxe organização e clareza ao sistema.
- O uso de data class garantiu uma modelagem de dados simples, segura e legível.
- A implementação de enum class assegurou consistência nos status dos pedidos.
- O sistema agora está mais manutenível, confiável e preparado para evoluções futuras.